OPEN SOURCE TRANSPARENCY FOR OFDM EXPERIMENTATION

Thomas W. Rondeau (CTVR, Trinity College Dublin, Dublin, Ireland, trondeau@vt.edu), Matt Ettus (Ettus Research, LLC., matt@ettus.com), Robert W. McGwier (CCR, IDA, rwmcgwier@gmail.com)

ABSTRACT

Many developments in communications systems begin as a theoretical model and then analyzed in a simulation environment such as Matlab. Often, these simulations do not provide the complexity of a real implementation where one must deal with real circuitry and channels. The open source GNU Radio platform offers an easy transition between the theoretical models and an affordable solution to test the models under real-world conditions. We present the concept of GNU Radio for this purpose here by discussing and analyzing a flexible OFDM transceiver. This design offers research and development of different methods of OFDM symbol transmission, reception, and synchronization. These concepts are currently being applied to develop waveforms such as WiMAX.

1. INTRODUCTION

GNU Radio is a free and open source software defined radio (SDR) project [1]. The purpose of this paper is to show how GNU Radio is useful in experimenting with and developing SDR techniques. Many problems faced in communication system design have multiple solutions while other new problems have solutions that are still in experimental stages of development. Some of these solutions have been developed and shown only mathematically and in simple simulations that ignore or miss certain problems experienced when used in real systems. Other times, different solutions to the same problems can have trade-offs in power cost and performance. In all of these cases, an open, transparent system for developing, testing, and comparing these systems can help advance our understanding.

GNU Radio is a solution to these problems. As an open source project, GNU Radio offers transparency to analyze, design, and distribute concepts in communications. We have developed a number of communications and signal processing blocks that can be used to build and test systems to analyze performance. Furthermore, GNU Radio runs in real-time and on-line and can be interfaced with RF hardware, which means we can go from experimentation to deployment in the same system.

In order to illustrate how GNU Radio offers these services to SDR development, we will discuss the implementation of an OFDM transceiver system. In particular, we have developed the transmit and receive chains in a modular way that enables discrete replacement of components so that different modulators, demodulators, and receivers can be developed and tested. We will explain the architecture and describe how to replace components in order to build new functionality such as modulator subcarrier mapping and timing and frequency synchronization methods. We have implemented three different synchronization blocks and will provide some results of their use to describe how to analyze and compare them.

This paper focuses on the implementation in GNU Radio, and so we will not present any more than the necessary description of OFDM and instead point to the relevant texts and papers. For a general introduction to OFDM, see [2]. To make the software and the information in this paper as useful as possible, all of the code, commands, and examples are distributed as part of the standard GNU Radio software package. The code is split between implementation in C++ and Python. The OFDM blocks written in C++ can be found in *gnuradio-core/src/lib/general* and are prefixed as *gr_ofdm_*. The Python blocks are hierarchical structures containing other hierarchical blocks and the C++ blocks and are found in *gnuradio-core/src/python/ gnuradio/blks2impl*.

2. TRANSMIT AND RECEIVE CHAINS

2.1. Transmit Chain

In OFDM, the data is mostly operated on in the frequency domain. Baseband data is manipulated using standard modulation techniques as a complex number and mapped to a vector which is then converted to the time domain using an IFFT. The vector is made up of N_{FFT} elements of which N_{OC} elements are used. When this is put through an IFFT of length N_{FFT}, the elements represent orthogonal subcarriers of which N_{OC} carry information. In general, these N_{OC} subcarriers occupy the middle subcarriers, leaving about $(N_{FFT}-N_{OC})/2$ subcarriers unused on either side (ignoring a ±1 if N_{FFT} is odd) as guard bands. The DC subcarrier is often removed to avoid problems caused by DC offsets.

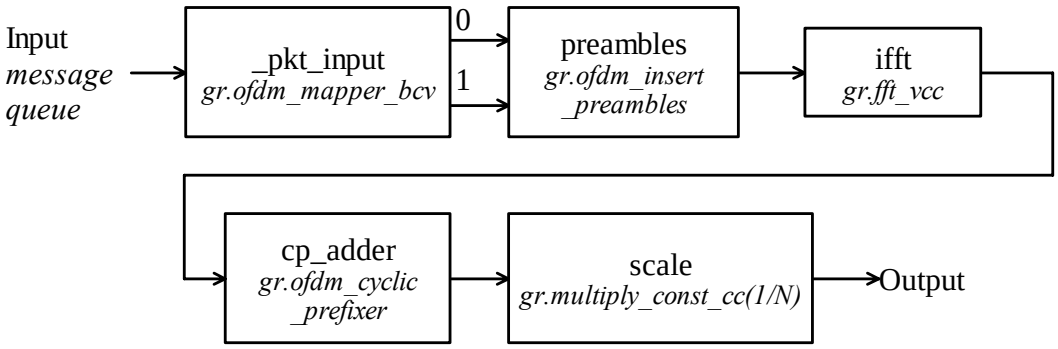

Figure 1. GNU Radio OFDM modulator block diagram (blks2impl/ofdm.py:ofdm_mod).

Following the IFFT, a guard time is usually inserted between symbols to protect against multipath and intersymbol interference (ISI). This is often done in the form of a cyclic prefix that takes the last *G* percent of the symbol and affixes it the front where *G* is often some fraction from 1/32 to 1/4.

Another important step in transmitting OFDM data is to guide the synchronization process at the receiver, which is usually done by adding a preamble of known data. We discuss this in more detail later.

Figure 1 shows the block diagram of the GNU Radio OFDM transmit chain. The blocks perform the baseband symbol mapping to the subcarriers, the insertion of preambles, the IFFT, and the addition of the cyclic prefix on each symbol. In the transmit chain, the *ofdm_mapper_bcv* function allows for some interesting modifications that are discussed in Section 2.3. The IFFT and cyclic prefix adder are standard operations while the preamble adder can take in a vector of preambles to give a large amount of flexibility in how the preambles are structured and how many are used.

The mapper block is currently implemented in the simplest way possible for basic functionality. The two subcarriers near DC are removed and the guard bands on the sides are evenly balanced. Data is modulated by taking the bit stream mapping the bits to a complex constellation that is passed in externally. The complex values are then sequentially inserted onto the subcarriers.

2.2. Receive Chain

The receive chain is more complex than the transmit chain and is split into the demodulator in Figure 3 with the receiver block in Figure 2 that performs the symbol timing and synchronization. After the channel filter, a synchronization routine is performed to find the symbol timing and fine frequency offset. The former clocks the symbols through the receive chain while the latter is used to adjust the numerically controlled oscillator (NCO) and a complex multiplier to adjust the frequency of the symbols. More about the synchronization mechanism will be provided in Section 3.

The OFDM sampler function uses the timing trigger to segment the incoming samples into the properly-timed symbols and cuts out the cyclic-prefix. The output is then demodulated through the FFT function that performs the inverse of the IFFT in the transmitter. The frame acquisition block completes the receiver synchronization by finding the final integer frequency offset and correcting for it as well as using the known data of the preamble to build an equalizer in the frequency domain. The output of this block is a derotated, equalized OFDM symbol. The OFDM frame sink block in Figure 3 takes the OFDM symbol data, demaps it into bits, and packs these bits into a frame to be passed out of the physical layer and up the stack. The frame sink uses the same constellation as the mapper function to perform the translation from the OFDM symbol to baseband bits.

2.3 Modifications

The mapper and frame sink (demapper) functions are where modifications can be made to enable different ways of realizing an OFDM system. These changes can be for experimental purposes or for actual use such as non-contiguous OFDM and constellation mapping.

Non-contiguous OFDM nulls certain subcarriers in order to manipulate the used spectrum by the system [3]. This concept has been discussed for use in dynamic spectrum access systems as a way to fill unused spectrum around narrowband users.

Another area of interest with more immediate applicability is in using different modulation constellations on different subcarriers. Based on the channel properties and interferers over all subcarriers, different modulations can provide a balance between data rate and BER performance [4].

Both of these techniques can be done through modifications of the mapper and frame sink functions. The routine for moving data onto the occupied subcarriers would control which subcarriers are used, and the constellation map provided to the function determines how data is mapped per subcarrier.

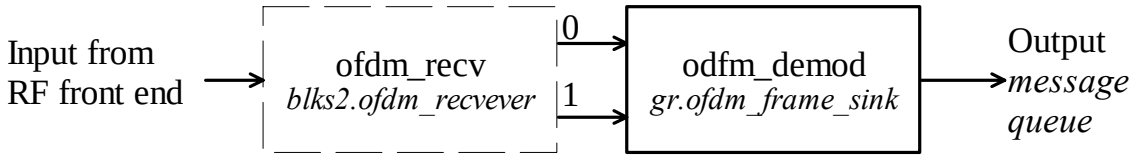

Figure 3. GNU Radio OFDM demodulator block diagram (blks2impl/ofdm.py:ofdm_demod). The ofdm_receiver block is a hierarchical block in Python and is shown in Figure 3.

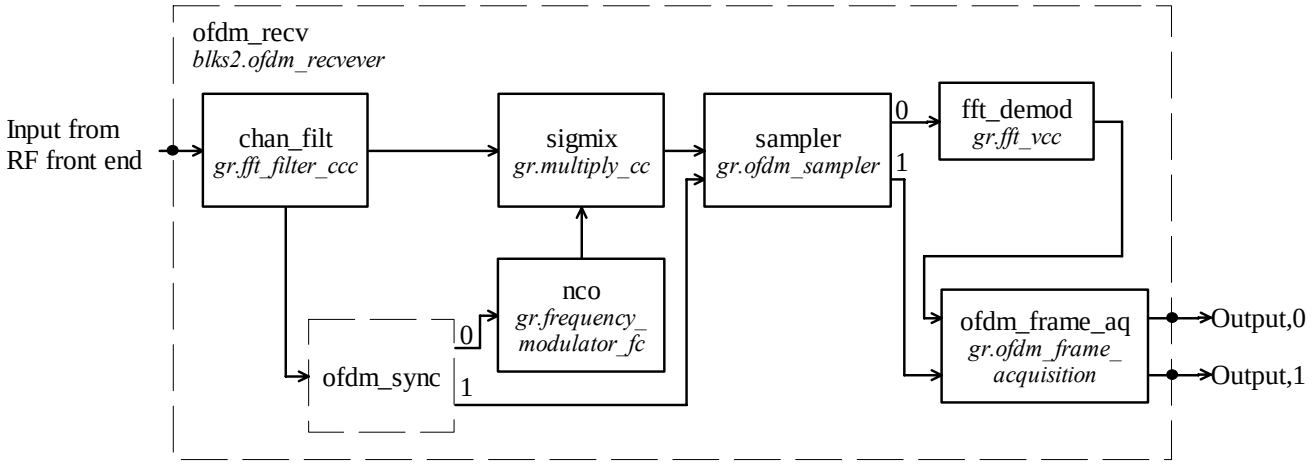

Figure 2. GNU Radio OFDM receiver (blks2impl/ofdm_receiver.py:ofdm_receiver). The ofdm_sync block is a hierarchical block in Python that can be replaced for different timing and frequency acquisition functions and techniques (see Section 4).

3. OFDM SYNCHRONIZATION

Like most OFDM systems in use, ours relies on the concept of prefixed known data at the start of a frame to provide information for synchronization at the receiver. The preamble is used by the receiver to find the start of the frame, estimate the frequency offset, and calculate an equalizer for the channel. Many standards synchronize based off a preamble that may be segmented into different segments of information for different parts of the receiver synchronization including the IEEE 802.16 standard [5]. We use a simple version of this, but since the preambles are passed from the user-space to the transmitter, the system allows for easy changes to this.

Our preamble is currently a repeated sequence in time over one full OFDM symbol by using a FFT property by inserting zeros into every other frequency bin. This is a well-known and used preamble concept and the basis of one of the most popular synchronization techniques by Schmidl and Cox [6]. Inserting two zeros between data symbols creates a three times repetition in the time domain that is part of the IEEE 802.16 OFDMA standard [5].

Equation 1 represents the OFDM symbol at the receiver after being put through a channel with complex taps H_n. Each subcarrier, n, is transmitted on a set frequency f_n but is received with some frequency offset specified by an integer part, f_Δ, and a fractional part, f_δ. The integer part specifies a number of subcarriers between the transmitted frequency and received frequency while the fractional part specifies the amount of frequency deviation within a subcarrier from the center. The complex envelope of additive white Gaussian noise is represented here by w_k.

$$y(t) = \left[\sum_{n=-N}^{N} H_n X_n \exp\left(j \frac{2\pi (f_n + f_\Delta + f_\delta) t}{T} \right) \right] + w_k \quad (1)$$

From equation 1, one of the most important functions of the receiver is estimating the frequency offsets, f_Δ and f_δ. The literature contains a number of ways in which these parameters are estimated, and to provide the most general interface we can, we have separated the synchronization tasks to a timing and fractional frequency synchronization block, which are usually done using the preamble symbol in the time domain, and an integer frequency synchronization and equalizer estimation in another block after the FFT. The order of synchronization is a bit counterintuitive since the fine frequency correction is done prior to the integer frequency correction. This is done because the timing and fine frequency correction methods are done in the time domain while the integer frequency correction is done in the frequency domain, and therefore after the FFT.

There are many proposed timing and fine frequency synchronization methods. In Section 4, we will present three of these that we have implemented in GNU Radio. We leave the specifics of how the synchronization is done to the literature. There are other synchronization techniques in the literature that are proposed for different purposes or with varying performance metrics in mind such as blind

synchronization methods for streaming services [7]. The intention of the design is to facilitate the introduction and testing of these different methods,

The integer frequency offset is found as part of the frame acquisition block that correlates the preamble in the frequency domain. The timing signal from the synchronizer block tells this block when the preamble is received, so it knows that the symbol it is working on is the preamble. The integer offset causes a symbol rotation, so the correlation is performed between the difference in the symbols on adjacent data subcarriers. The subcarrier offset that maximizes the correlation is the integer frequency offset. This value is then used to fully derotate the symbol through equation 2 where the each input symbol, $y[i]$, is multiplied by the rotation of the integer frequency offset and the ratio of the cyclic prefix length, N_{cp}, to the FFT length, N_{FFT}.

$$\hat{y}[i] = y[i]\exp\left(j2\pi f_{\Delta} i \frac{N_{cp}}{N_{FFT}}\right) \quad (2)$$

The equalizer is calculated by using the known symbol or the preamble and the received preamble. Because of the nulled subcarriers used for the symbol repetition, the equalizer is determined for every second subcarrier as $H[i] = x[i] / \hat{y}[i]$ where $x[i]$ represents the known value of the preamble in subcarrier i. The other subcarriers are calculated by simply interpolating between these points.

4. COMPARING SYCHRONIZATION METHODS

In this section, we provide a brief review of the three synchronization techniques developed for use in GNU Radio. For a complete understanding of synchronization, see [8] and the individual papers behind each technique.

The synchronizers are all located in Python source directory and are prefixed as *ofdm_sync_*. They are called from *ofdm_receiver.py* and a simple *if* statement selects which method to use based on the suffix of the filename. These techniques are implementations of the "ofdm_sync" hierarchical block of Figure 3. We do not show their block diagrams here due to space constraints.

4.1 Schmidl and Cox

The first and default synchronizer is based on the Schmidl and Cox method [6] (*ofdm_sync_pn.py*). The basis of this method correlates against a repeated PN code in the preamble and creates a timing trigger from it. The correlation also produces a fine frequency estimation within ±1 subcarrier spacing. The integer portion is calculated later in the *ofdm_frame_acquition* block.

4.2. van de Beek

The van de Beek method [9] (*ofdm_sync_ml.py*) correlates against the cyclic prefix on every symbol, producing a timing and frequency estimate each time. Frequency estimation is good to ±0.5 subcarrier spacing.

Referring to Figure 3, each synchronization method is expected to produce a fine frequency correction signal used to adjust the frequency of the incoming signal as well as a timing signal that indicates the start of the frame. This method produces a timing signal for every received symbol and not just once per frame. When an OFDM symbol is received with a cyclic prefix, the circular convolution property of the symbol allows full demodulation anywhere in the cyclic prefix up to the first sample of the symbol. This holds true as long as a full T samples are received. Ideally, the timing should place the start of the symbol exactly at the end of the cyclic prefix to avoid any ISI. However, the timing algorithms in real systems do not necessarily get this kind of accuracy. If the timing starts inside the cyclic prefix, the data can still be recovered but with a phase shift proportional to the timing position. The *ofdm_frame_acquisition* block uses the known preamble to calculate an equalizer that can then correct for the phase shift along with the channel properties. If during the reception of the frame, the timing is adjusted by even a single sample, the timing offset introduces a new phase difference that the equalizer can no longer correct.

To protect the receiver from this, we have added a correlation against the preamble and use this as the output timing trigger. While this provides the proper timing, the correlation procedure removes the ability to handle large frequency offsets since the correlation peak degrades as the offset increases. A coarse frequency estimation would be required prior to this if this system were to be used in a situation where a large frequency offset is expected.

4.3. Tufvesson

The Tufvesson method [10] (*ofdm_sync_pnac.*py) is a modification of the Schmidl and Cox method that adds a cross-correlation against the known PN sequence of the preamble before going through the time-delayed correlation. This produces a stronger timing signal at the end of the preamble and reduces ambiguity in the timing. However, like the correlation added to the van de Beek method, the cross-correlation here affects the maximum frequency offset that can be detected.

5. DISCUSSION

To illustrate both the successful implementations of these techniques as well as show some of the differences, we present here few output results of the different synchronizers. The tests are run using the *benchmark_ofdm.py* simulation test script that is a part of the GNU Radio distribution package and can be found in the source tree at *gnuradio-examples/src/python/ofdm/*. The scripts *benchmark_ofdm_tx.py* and *benchmark_ofdm_rx.py*

are then used to transmit and receive the signals using the Universal Software Radio Peripheral (USRP) [11].

The script is run for each of the three synchronizers with the command-line arguments:

./benchmark_ofdm.py --log --frequency-offset=0.3 --tx-amp=2 -s 1500

This sets the frame length (-s) to 1500 bytes, a fractional frequency offset of 0.3 subcarriers, sets a specific amplitude value, and logs all debug data to files. By default, the following values are used by the script.

Table 1. OFDM Simulation Parameters

Parameter	Value
FFT length	512
Used subcarriers	200
Cyclic prefix length	128
SNR	30 dB
Timing offset	0
Multipath channel	Off
Modulation	BPSK
Symbols per frame	62

GNU Radio distributes a few useful plotting tools to visualize the output data using the open source software projects Python, Scipy, Numpy, and Matplotlib. One plotting tool is distributed in the OFDM examples directory that was built specifically to analyze the OFDM system and is called *gr_plot_ofdm.py*. Another suite of plotting utilities are located in *gr-utils/src/python/* that are all prefixed *gr_plot_* and can plot character, integer, floating point, and complex (as I&Q) data, plot the FFT of floating or complex data, and plot the constellation of complex data.

To show the performance of the synchronizers, we use the OFDM-specific plotting tool which plots four views of the received symbols. The upper left plot shows the constellation of the received baseband symbols. The black dots represent the actual data symbols while the white dots represent the data after being passed through a decision-directed equalizer to correct for phase tracking offsets. This plot is useful in understanding the error performance of the system as the symbols are tracked through a full frame. The middle two carriers are nulled out to compensate for any DC offset and show up at the origin of this plot.

The upper right plot shows the unequalized angle across all FFT bins where the middle occupied carriers carry the data. This plot shows the effects of timing offsets in the synchronizer through the angle of the occupied carriers.

The equalized phase in the bottom left corner shows the compensation for any channel affects as well as the linear phase offset due to timing error. We did not use a multipath channel in this simulation to make the results more clear. A simple simulated multipath channel can be used by adding "--multipath-on" to the command-line. We have not modeled any specific multipath channels yet, so this just applies an FIR filter with preset taps.

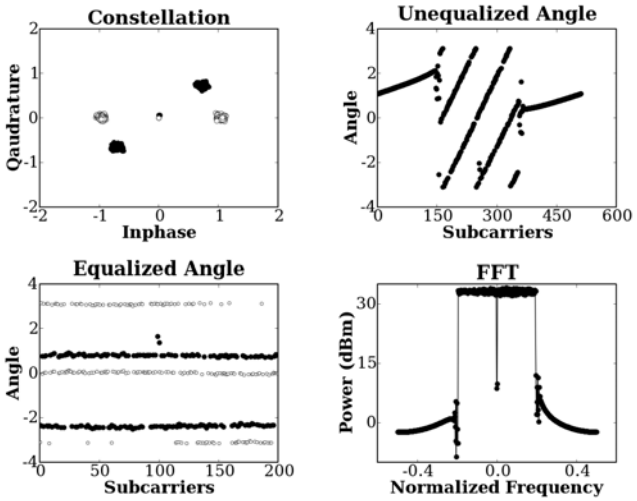

Figure 4. Results of Schmidl and Cox synchronization.

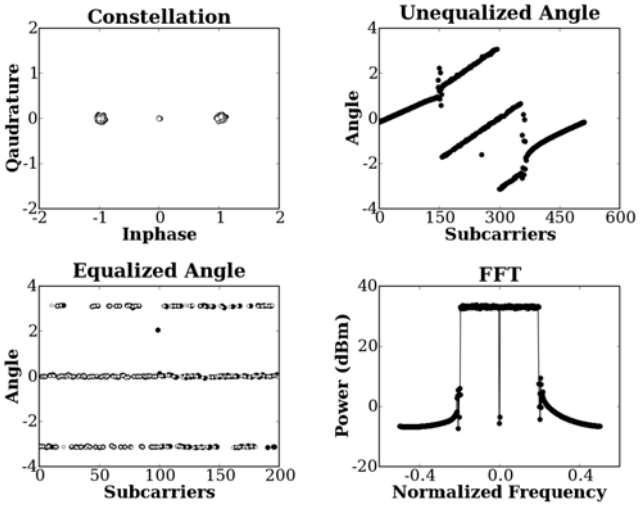

Figure 5. Results of van de Beek synchronization.

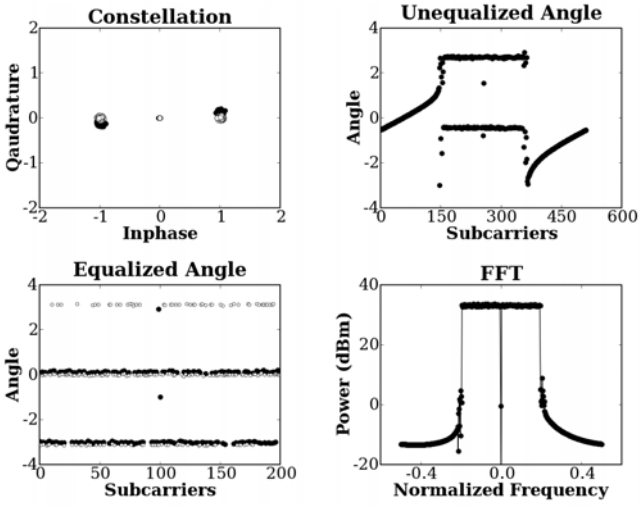

Figure 6. Results of Tufvesson synchronization.

From these plots, we can see differences in the synchronizer's performance. The Schmidl and Cox and the

van de Beek methods have issues when looking for the exact timing point due to digitization error, which causes the linear phase shift in the unequalized angle plot. This is not a problem because the equalizer corrects for the phase shift. The Tufvesson method provides a more accurate timing metric and therefore a flat phase response even before the equalizer is applied.

Both the Schmidl and Cox and the Tufvesson methods have another interesting effect. Each of these plots was captured on the final symbol in the frame. The black points in the constellation diagram and the equalized phase shows a phase rotation that occurs throughout the frame. This occurs because of the ambiguity in the estimation of the fractional frequency offset, which builds a small rotation into each symbol that builds up over the length of the frame. The van de Beek method is the maximum likelihood receiver and calculates the frequency much more accurately. Furthermore, because the van de Beek method performs the frequency estimation once per symbol as opposed to once per frame, each symbol is independently corrected and the frequency is tracked throughout the frame.

5. CONCLUSIONS AND DEVELOPMENTS

The current OFDM implementation provides the basic properties required to both transmit and receive data through stationary channels. The system corrects for multipath, frequency and timing offsets, and can transfer data with a variable number of subcarriers, cyclic prefix length, and subcarrier modulation. There are more properties that we are currently making progress on implementing. One important OFDM property is the use of pilot tones. Pilot tones are reserved subcarriers that are modulated with a known symbol used by the receiver to correct for changes in the channel, which is required for dynamic channels. The pilots are often implemented as a lattice that can be multidimensional across both frequency and symbol. The addition of pilots is already in progress.

IEEE 802.16 WiMAX, the developing wireless standard for highspeed communications, uses OFDM and OFDMA. It is a current goal of GNU Radio to provide the necessary capabilities and flexibility to realize all of the variables of the WiMAX standard. Figure 7 shows the use of QAM64 modulation on all of the subcarriers to demonstrate some of the other capabilities of the current system. Other, intermediate QAM modulations between QAM64 and BPSK are obviously also supported.

At the Laboratory for Telecommunications Sciences, University of Maryland, a WiMAX network is being deployed for academic experimentation. It will also be an official industry wide experimentation facility in conjunction with several WiMAX working groups. LTS is using GnuRadio, supporting it for more rapid development, and implementing WiMAX 802.16e waveforms.

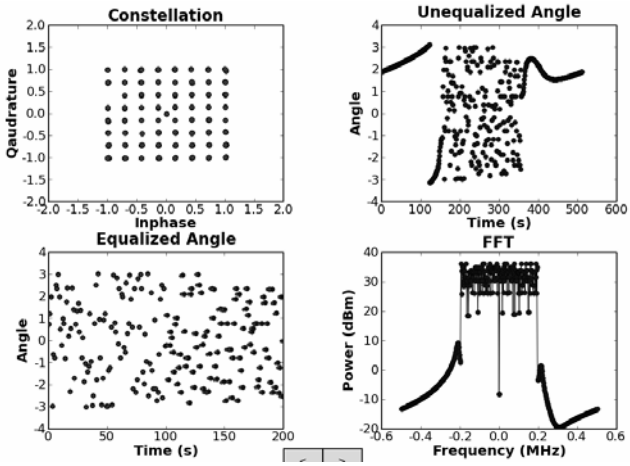

Figure 7. Results of using QAM64 subcarrier modulation.

GNU Radio is a platform for both experimentation and development of communication systems. This paper has shown how we have developed OFDM capabilities for GNU Radio through the implementation details and experimentation with different synchronization algorithms. Through this, we have shown how we are developing GNU Radio capabilities as well as building, improving, and disseminating knowledge.

6. REFERENCES

[1] GNU Radio, http://www.gnuradio.org/, 2008.

[2] J. G. Andrews, A. Ghosh, and R. Muhamed, *Fundamentals of WiMAX: Understanding Boradband Wireless Networking*, Prentice Hall, 2007.

[3] T. A. Weiss, F. K. Jondral, "Spectrum pooling: an innovative strategy for the enhancement of spectrum efficiency," *IEEE Communications Magazine*, vol. 42, no. 3, pp. S8-14, 2004.

[4] S. Sampei, H. Harada, "System Design Issues and Performance Evaluations for Adaptive Modulation in New Wireless Access Systems," *Proceedings of the IEEE*, vol. 95, no. 12, pp. 2456-2471, Dec. 2007.

[5] IEEE, "IEEE 802 Part 16: Air Interface for Fixed Broadband Wireless Access Systems," IEEE Std 802.16-2004, 2004.

[6] T. M. Schmidl, D. C. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Trans. Communications*, vol. 45, no. 12, pp. 1613-1621, Dec. 1997.

[7] S. L. Talbot and B. Farhang-Boroujeny, "Spectral modelling and low-complexity blind carrier frequency tracking in OFDM," in /Proc. ICC 2006, /vol. 7, pp. 2917-2922, Jun. 2006.

[8] P. H. Moose, "A technique for orthogonal frequency division multiplexing frequency offset correction," *IEEE Trans. Communications*, vol. 42, no. 10, pp. 2908-2914, Oct. 1994.

[9] J. J. van de Beek, M. Sandell, P. O. Borjesson, "ML estimation of time and frequency offset in OFDM systems," *IEEE Trans. Signal Processing*, vol. 45, no. 7, pp. 1800-1805, Jul. 1997.

[10] F. Tufvesson, O. Edfors, and M. Faulkner, "Time and Frequency Synchronization for OFDM using PN-Sequency Preambles," IEEE Proc. VTC, 1999, pp. 2203-2207.

[11] Ettus Research, LLC., http://www.ettus.com, 2008.